Duplicate Detection On the Web

Finding Similar Looking Needles...

...In Very Large Haystacks

Sergei Vassilvitskii

Yahoo! Research, New York

November 15, 2007

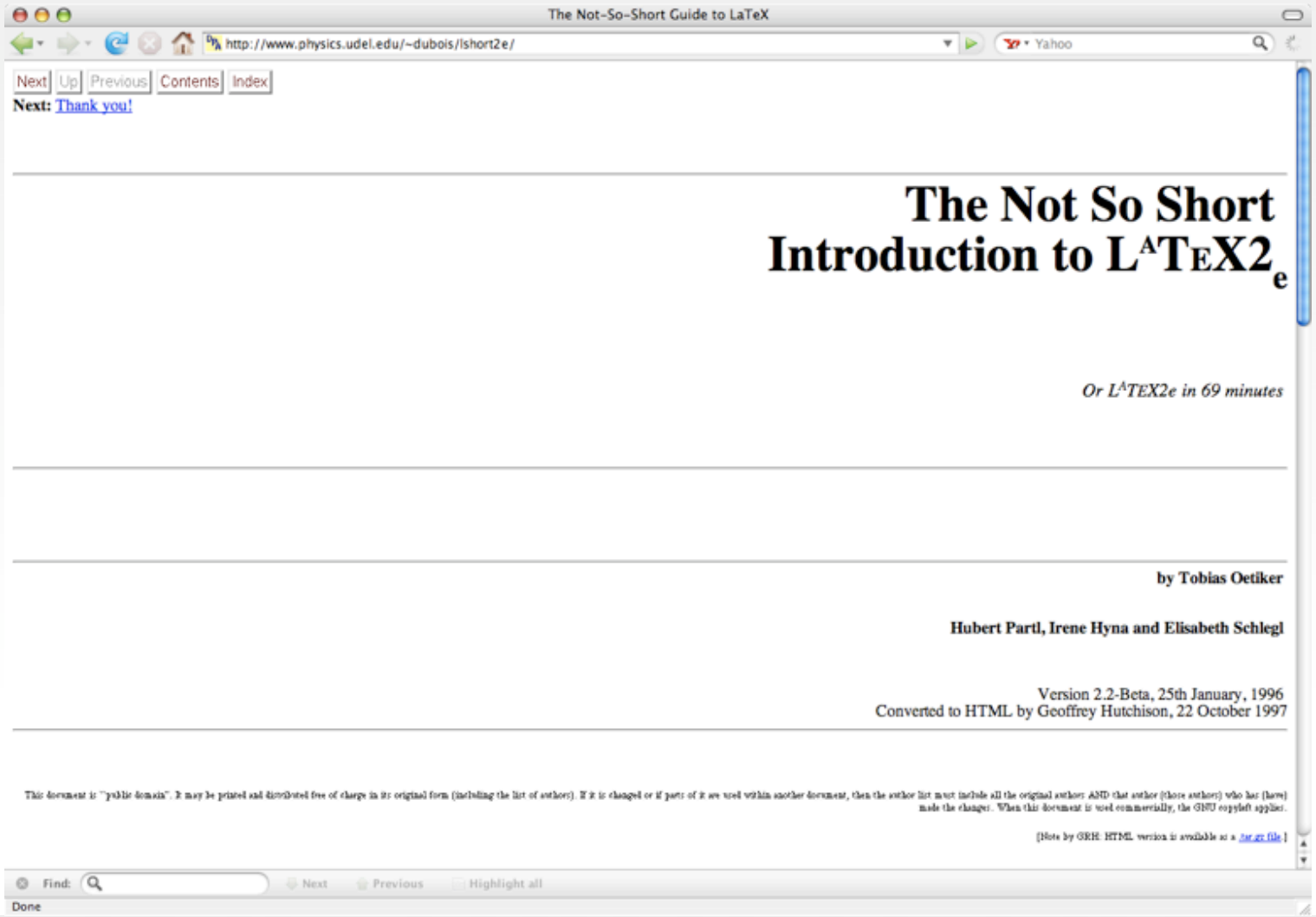
Duplicate Detection

- Why detect duplicates?
- Conserve resources
  - reduced index size - less memory, faster computations, etc.
- User Experience
  - Diversity in Search Results
  - 25-40% of the web is duplicate
    - Identical reviews with different boilerplate
    - mirrors - e.g. unix man pages
    - SPAM sites
    - etc etc

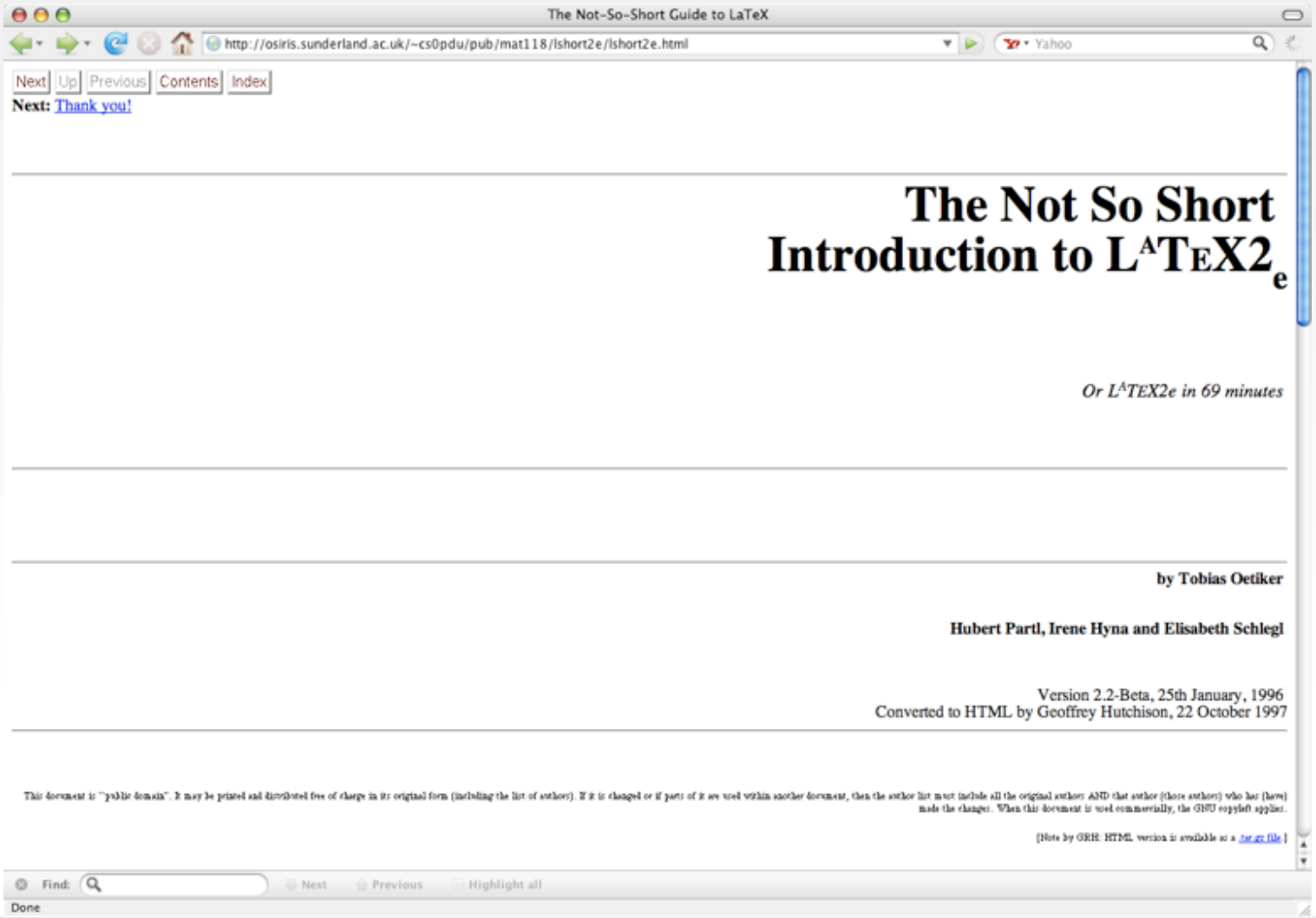
What's a Duplicate?

- Easy: exact duplicates

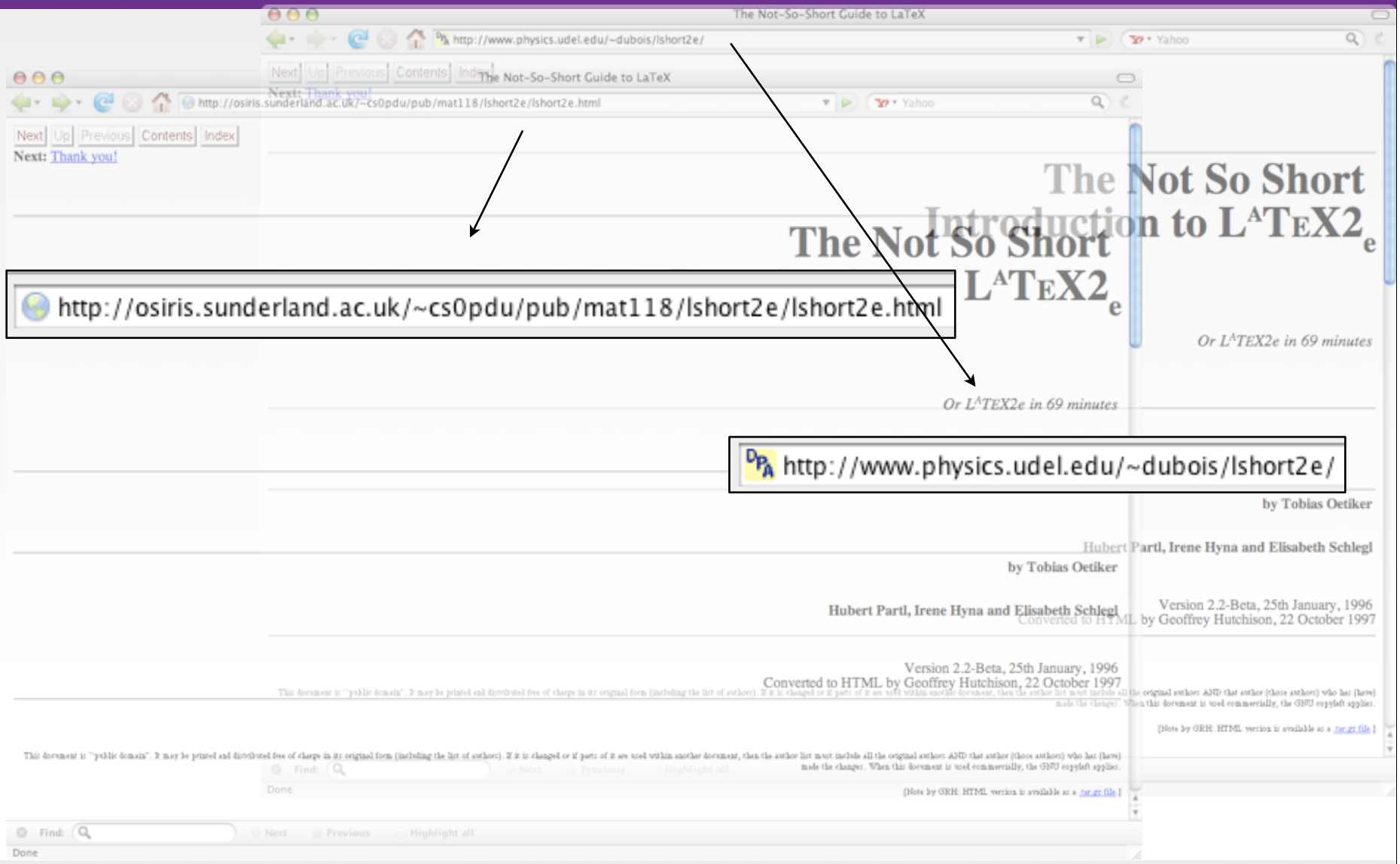
Exact Duplicates



Exact Duplicates



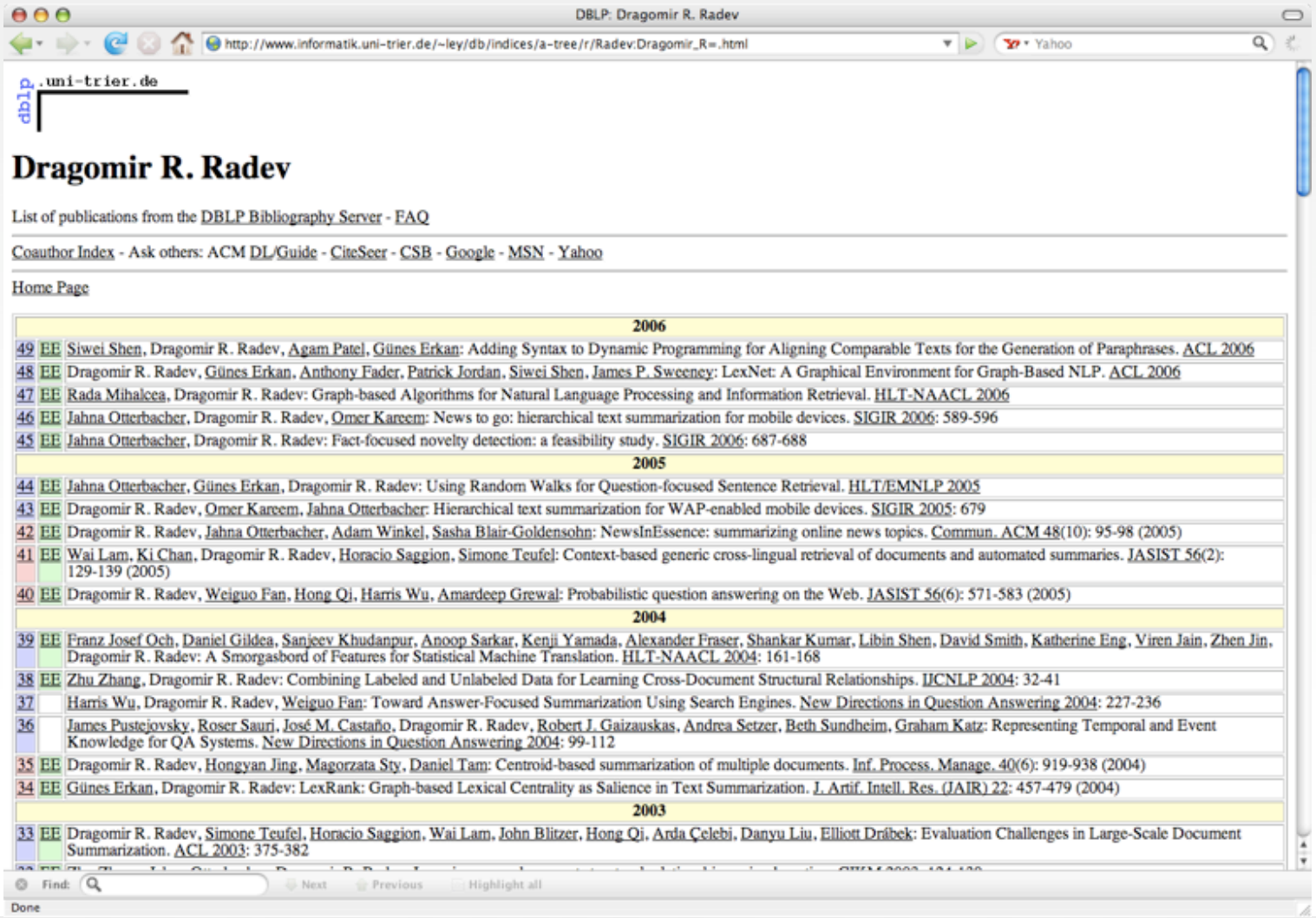
Exact Duplicates



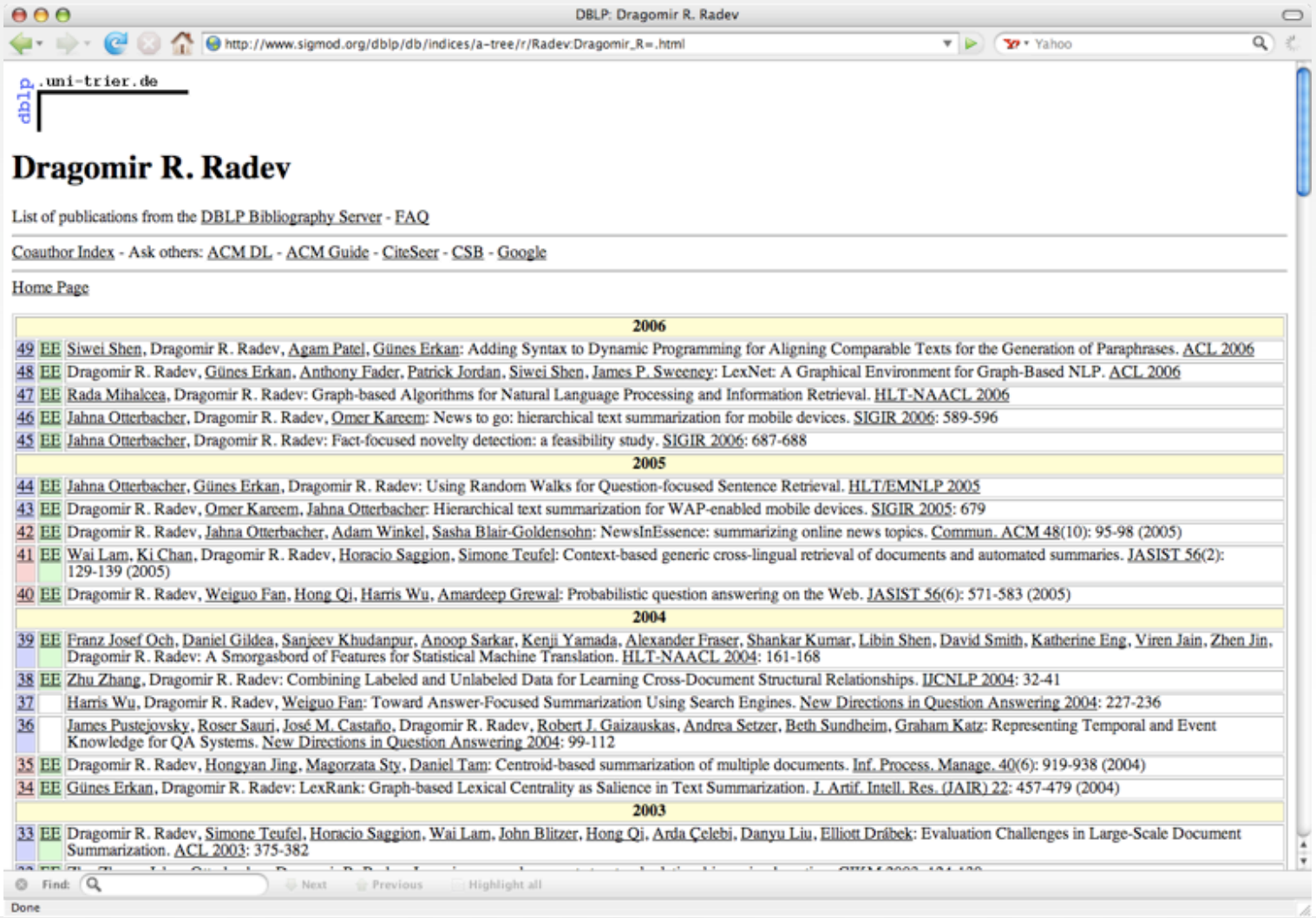
What's a Duplicate?

- Easy: exact duplicates
- Still easy: Different dates / signatures

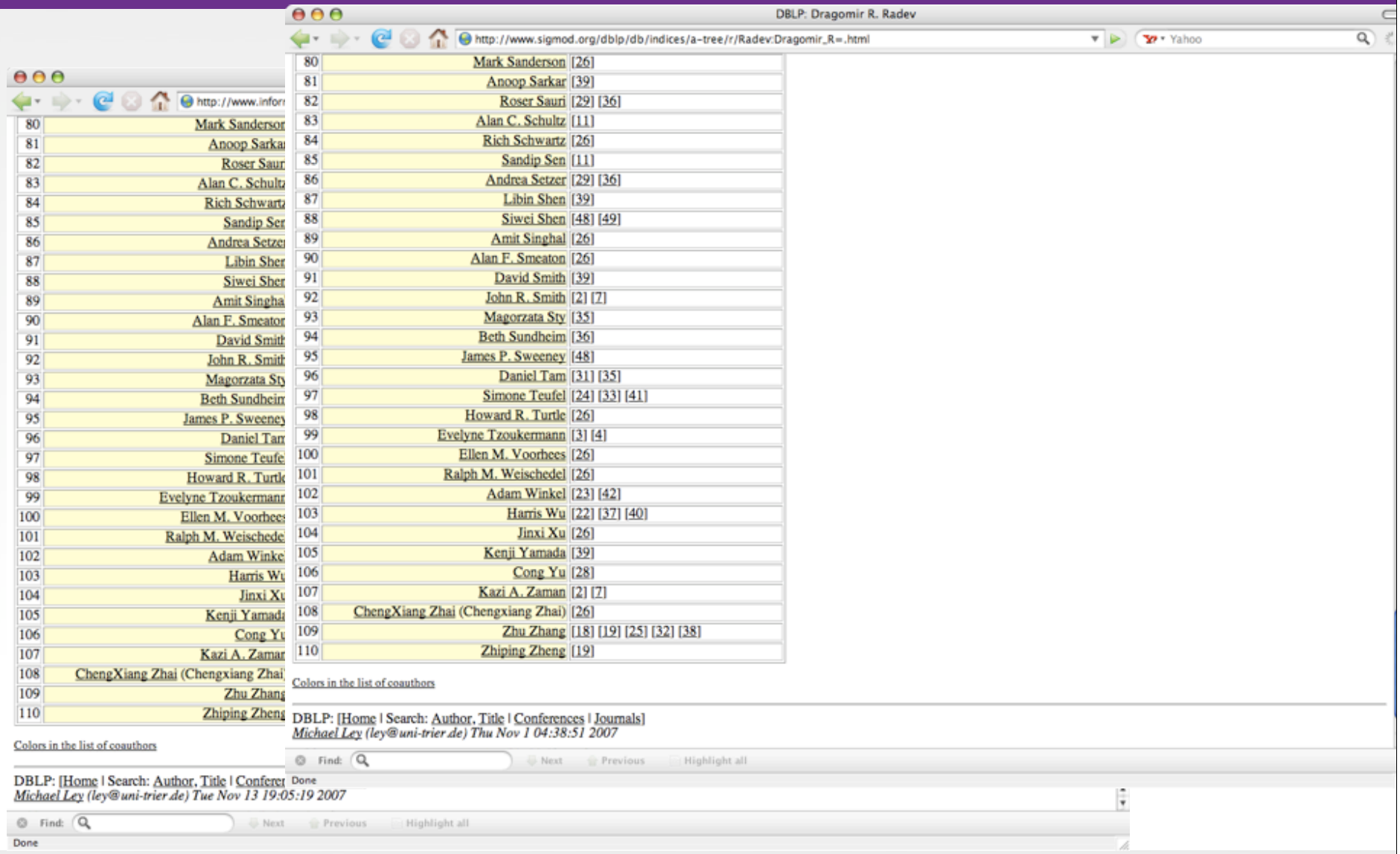
Almost Duplicates



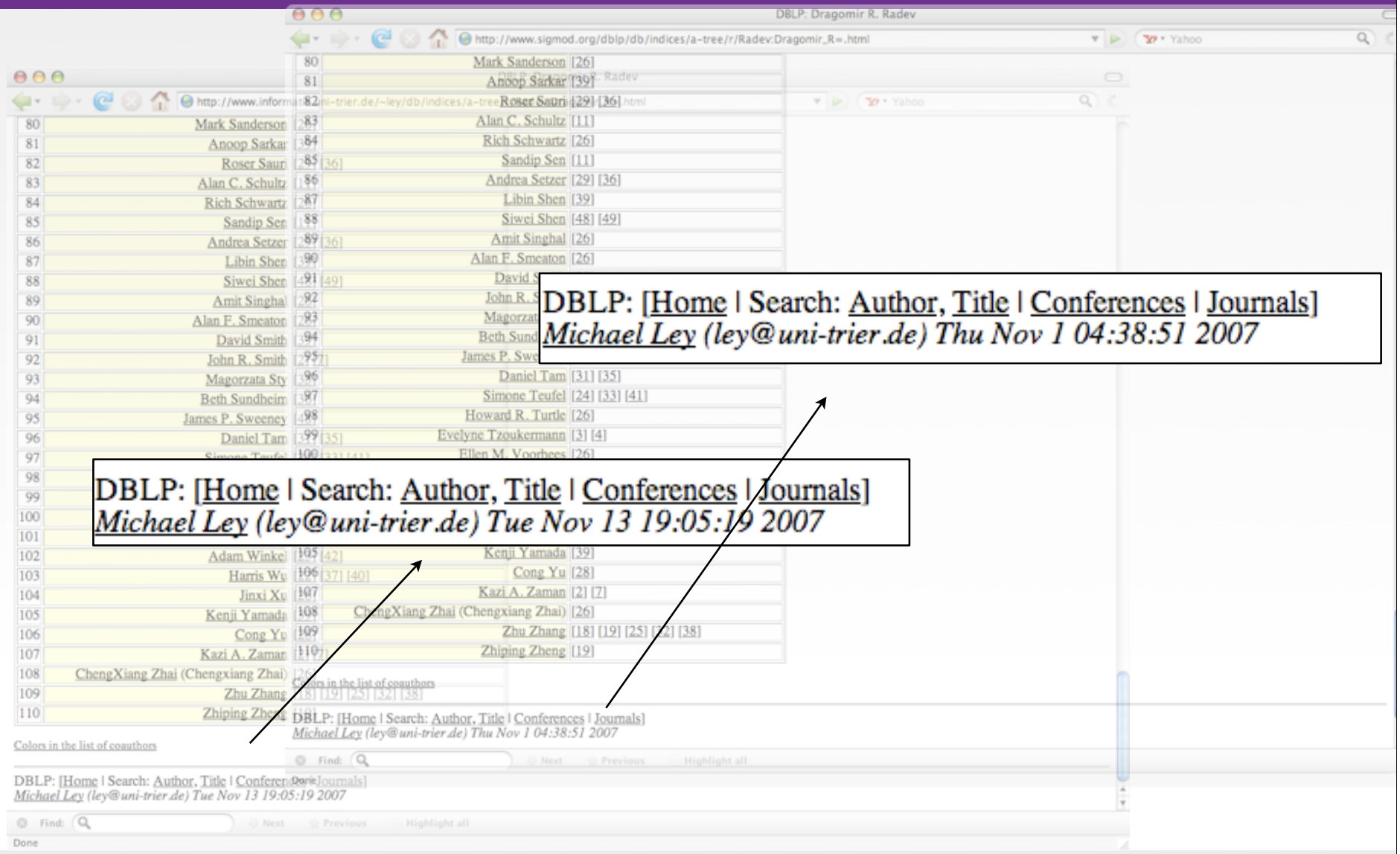
Almost Duplicates



Almost Duplicates



Almost Duplicates



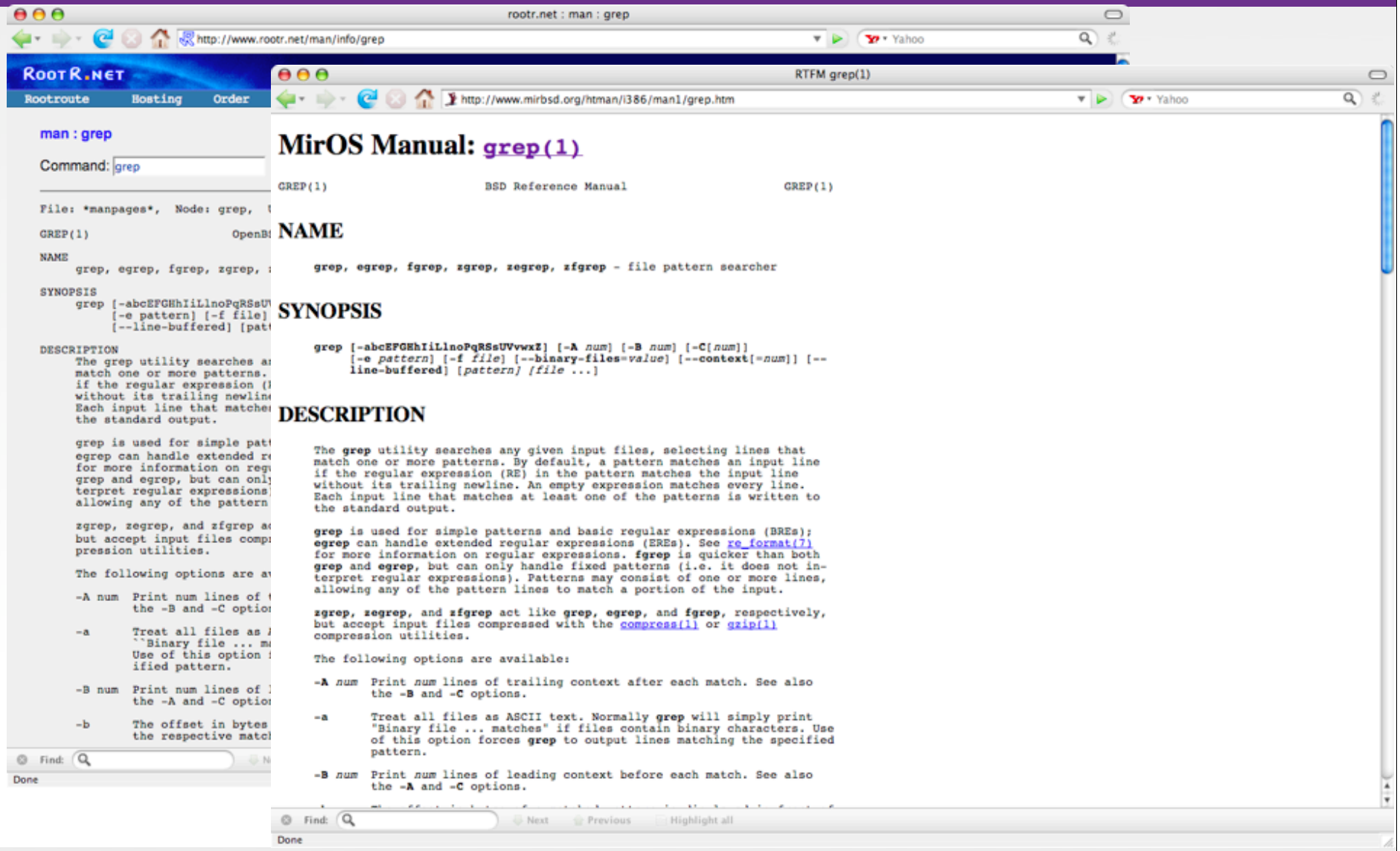
What's a Duplicate?

- Easy: exact duplicates
- Still easy: Different dates / signatures
- Harder: Slight edits, modifications

is this a duplicate?

is this a duplicate?

is this a duplicate?



What's a Duplicate?

- Easy: exact duplicates
- Still easy: Different dates / signatures
- Harder: Slight edits, modifications
- Hardest: Different versions, updates, etc.

Similarity

- Tokens - trigrams in a document:

```
Once upon a midnight dreary, while I pondered weak and weary
```

```
Once upon a
     upon a midnight
          a midnight dreary
            midnight dreary  while
```

- Represent a document as a set:

```
{Once upon a, upon a midnight, a midnight dreary, ... }
```

- Similarity (A,B) = $\frac{|A \cap B|}{|A \cup B|}$

Similarity

- A: "Once upon a midnight dreary, while I pondered"

  { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered }

- B: "Once upon a time, while I pondered"

  { Once upon a, upon a time, a time while, time while I, while I pondered }

Similarity

- A: "Once upon a midnight dreary, while I pondered"

  { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered }

- B: "Once upon a time, while I pondered"

  { Once upon a, upon a time, a time while, time while I, while I pondered }

- Overlap: $|A \cap B| = 2$

Similarity

- A: "Once upon a midnight dreary, while I pondered"

  { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered }

- B: "Once upon a time, while I pondered"

  { Once upon a, upon a time, a time while, time while I, while I pondered }

- Overlap: $|A \cap B| = 2$
- Total: $|A \cup B| = 9$

Similarity

- A: "Once upon a midnight dreary, while I pondered"

  { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered }

- B: "Once upon a time, while I pondered"

  { Once upon a, upon a time, a time while, time while I, while I pondered }

- Overlap: $|A \cap B| = 2$
- Total: $|A \cup B| = 9$
- Similarity 0.22

Similarity

- Recall Sim(A,B) = $\frac{|A \cap B|}{|A \cup B|}$
  - Also known as the Jaccard similarity
- Is this a good similarity measure?
  - Yes: Simple to describe, easy to compute
  - No: Ignores repetition of trigrams:
    - e.g. "`a rose is a rose`" and "`a rose is a rose is a rose`" have 100% similarity.
    - Bad at detecting small but semantically important edits

Outline

- Motivation
- Algorithms
- Evaluation
- Open Problems

Algorithms

- Hashing:
  - Perfect for detecting exact duplicates. Doesn't work for near dupes.
- Edit distance
  - Perfect for detecting near dupes, does not scale.

Scale

- The size of the index is claimed to be 4B-16B pages
- Exact estimation is an active research topic
  - Also, bigger doesn't always mean better
- Safe to assume at least 1B pages in index.

Algorithms

- Hashing:
  - Perfect for detecting exact duplicates. Doesn't work for near dupes.
- Edit distance
  - Perfect for detecting near dupes, does not scale.
  - Compare every time.
- Shingling
  - We will store 48 bytes per page, detect near duplicates without examining all 1B pages.

Shingling Idea

- Computing Jaccard similarity is still expensive.
- Idea: summarize each document in a short sketch.
- Estimate the similarity based on the sketches.

- Algorithm due to Broder et al. (WWW '97), used in the Altavista search engine and all search engines since.

Algorithm

- Take a hashing function, H. Hash each shingle:

  { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered }

  { 357, 192, 755, 123, 987, 345 }

- Store the minimum hash: {192}

  { Once upon a, upon a time, a time while, time while I, while I pondered }

  { 357, 143, 986, 743, 345 }

- Store the minimum hash: {143}

Algorithm

- Repeat many times with different hash functions

	hash-1	hash-2	hash-3	hash-4
doc 1:	192	155	187	255
doc 2:	143	179	187	155

- Similarity - Percentage of times hashes agree

- SIM(doc 1, doc 2) = 1/4

Why Min-Hash

- Theorem:

$$Prob\left[\texttt{Min-Hash}(A) = \texttt{Min-Hash}(B)\right] = \frac{|A \cap B|}{|A \cup B|}$$

- Therefore:
  - % of time the hashes agree ~ Sim(A,B)

Why Min-Hash

- Proof: Look at the elements:

  { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered, upon a time, a time while, time while I}

- One of these will have the minimum hash value.
- The two minima are the same if the corresponding trigram appears in both documents.
- Min-hashes are equal if either of { Once upon a, while I pondered } hashes to the minimum value.

Sketches

- So a sketch for a document is a collection of minimum hashes.
  - In practice, use 84 hash functions.
- sketch = `{ 192, 155, 187, 255, ..., 101 }`

- Summarized each page in 672 bytes (84 8 byte values).

Sketches

- To compare two documents, look at the percentage of min-hashes that agree.
- Problem: Full Pairwise comparison
  - 10^9 pages by 10^9 pages by 10^2 hashes = 10^{20} operations.
- But we have many fast computers!
  - 10^9 operations / second * 10^4 machines would still require 10^7 seconds - roughly 4 months.

Super Shingles

- Problem: Doing all pairwise comparisons still too expensive.
- Solution: Since we care about only high similarity items, recurse:
- sketch = { 192, 155, 187, 255, 345, 171, 877, ... , 101 }
- Group into non overlapping super-shingles:

{192, 155, 187, 255}

{345, 171, 877, ...}

{..., 101}

- Hash each super-shingle: {1011, 6543, ..., 7327}
- Only compare documents that agree on super-shingles.

Super Shingles

	S-Shingle 1	S-Shingle 2	S-Shingle 3
Doc 1	1011	6543	7327
Doc 2	4523	5498	8754
Doc 3	5487	5498	8754

Super Shingles

	S-Shingle 1	S-Shingle 2	S-Shingle 3
Doc 1	1011	6543	7327
Doc 2	4523	5498	8754
Doc 3	5487	5498	8754

- Declare Doc2 and Doc3 to be 2-similar.
- In practice - store the above table sorted by different columns. Only compare against neighboring rows.

Super Shingles

- Store the super shingle table sorted by columns

	S-Shingle 1	S-Shingle 2	S-Shingle 3
Doc 1	1011	6543	7327
Doc 2	4523	5498	8754
Doc 3	5487	5498	8754
Doc 4	8766	1258	6255

	S-Shingle 1	S-Shingle 2	S-Shingle 3
Doc 4	8766	1258	6255
Doc 2	4523	5498	8754
Doc 3	5487	5498	8754
Doc 1	1011	6543	7327

- Only compare adjacent rows instead of all pairs

Summary

- Text = Once upon a midnight dreary, while I pondered ...
- Trigrams = { Once upon a, upon a midnight, a midnight dreary, midnight dreary while, dreary while I, while I pondered, ... }
- Hashes(1) = { 357, 192, 755, 123, 987, 345, ... }
- Min Hash(1) = { 192 }
- Hashes(2) = { 132, 345, 487, 564, 778, 120, ... }
- Min Hash(2) = { 120 }
- ..
- Min Hash(84) = { 101 }
- Sketch = { 192, 120, ..., 101 }
- Super Shingles = {1011, 6543, 7327, 5422, 8764, 2344}
- Similar if exact overlap on 2 or more super shingles.

Outline

- Motivation
- Algorithms
- Evaluation
- Open Problems

Evaluation

- Due to Henzinger, SIGIR '06
- Start with 1.6B webpages - 46M hosts, on average 35 pages per host.
- Remove exact duplicates (around 25%)

- Impossible to check recall of the algorithms (Why?)
- To check precision: sample roughly 2000 pairs returned as duplicates and evaluate

Evaluation

- Pairs are decided to be near duplicate if:
  - Text differs by timestamp, visitor count, etc.
  - Difference is invisible to the visitor
  - Entry pages to the same site
- Not near duplicate if:
  - Main items are different (e.g. shopping page for two different items, but with identical boilerplate text)

Evaluation

- Undecided:
  - Prefilled forms with different values
  - A different ‘minor’ item - e.g. small text box
  - Pairs that could not be evaluated (e.g. english speaking evaluator looking at two Korean pages)

Results

Precision on the ‘duplicate’ set.

	Pairs	Correct	Incorrect	Undecided
All	1910	0.38	0.53	0.09
2-sim	1032	0.24	0.68	0.08
3-sim	389	0.42	0.48	0.1
4-sim	240	0.55	0.36	0.09
5-sim	143	0.71	0.25	0.06
6-sim	106	0.85	0.05	0.1

Outline

- Motivation
- Algorithms
- Evaluation
- Open Problems

Open Problems

- The devil is always in the details:
  - Obtaining text from raw HTML is not as easy as it sounds
    - What to do with IMG ALT text? Targets of links? etc.
  - Large boilerplate text with few seemingly minor differences
- New Challenges
  - Dynamic content?
  - Flash, Ajax, and other not easily indexable content

References

- Broder, Glassman, Manasse, Zweif. Syntactic clustering of the web. WWW '97.
- Fetterly, Manasse, Majork. Detecting phrase level duplication on the World Wide Web. SIGIR '05.
- Henzinger. Finding near duplicate Web pages: a large scale evaluation of algorithms. SIGIR '06.

Thank You!